Spring 2008 Final Report

May 17th 2008

Seong-hee Lee (Emily)
sl486@cornell.edu

Abstract

The goal for this semester's work was to test future electronics components. Future electronics sub-team's job was to design a nervous system of the walking robot. The nervous system consists of one mother and several daughter boards communicating with certain data protocol. For spring 2008 semester, my task was to choose optimal electrical components and data protocol by testing several CAN bus transceivers under different conditions. Additionally, the motor control board for the next robot was tested to determine the thermal characteristic of certain H-bridge, which will possibly be used in the next robot.

Introduction

1. Purpose of CAN Bus Testing

CAN bus is widely used for relatively low speed system that contains analog data, such as automobiles. Similar to cars, walking robot contains lots of analog data; however, unlike cars, walking robot requires high-speed system with low energy consumption. The purpose of the test was to measure how reliable and energy efficient CAN bus protocol is with different components (transceivers, termination resisters, wire schematics, and wire lengths) at high transmission rate (baud rate up to 4Mbps).

2. Purpose of H-Bridge Testing

H-bridge is the central component of motor control and also susceptible to heat. Therefore it is important to test how much current and voltage the H-bridge can endure.

Test Procedure and Results

1. CAN Bus Testing

1.1. Preparation

1.1.1. Test Board Design

In order to test CAN bus communication, I designed a test board in order to test 4 different transceivers: MAX 3057 (Maxim), MAX 3051 (Maxim), SN65VHD230 (Texas Instruments), and SN65VHD233 (Texas Instruments). The schematic and the board design can be found in Appendix A. The board is divided into four quarters and each quarter has two transceivers of a kind and one 6-pin connector. It is designed so in order to test two possible wire configurations: one with a DC wire in between two CAN data transmission wires, and one without. Since one transceiver (MAX 3057) needs 5V power supply, while others need 3.3V, four quarters have separate voltage supplies while sharing the same ground. Also, transceivers and connectors are connected to big vias (holes to solder wires on) in order to make the wire configuration modifiable after the board is made.

1.1.1.a) Populating the Board

In order to solder small components effectively, I used the reflow soldering method. There was no shorted trace or malfunctioning components after the soldering suggesting that it is a reliable method despite some doubts about using a toaster oven for reflow soldering.

1.1.1.b) Debugging the Board

After the population of the board, a mistake in the schematic was found. For SN65VHD233, pin 5 is supposed to be open or connected to ground in order for the transceiver to transmit data. It required lifting pin 5 of the component and rewiring to fix the problem.

1.1.2. Microcontroller Programming

Phillips LPC 2129 (or some other LPC 21XX) was selected for the microcontroller of the electronic nervous system. Appendix A.1 contains the complete code. The detail about programming is as below:

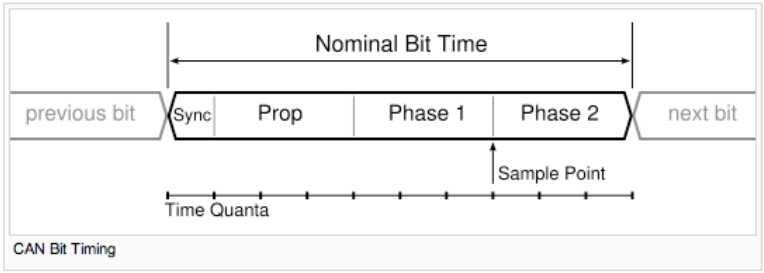
1.1.2.a) Processor Initialization
("Include Files" to "Initialize PLL" section of the code)

This part of the code was mostly borrowed from Sam's LPC 21XX initialization code. The main point of this part is to set phase-locked loop (PLL) in order to set the processor clock, which is the reference of the peripheral clock.

1.1.2.b) CAN Bus Initialization
("init_CAN2" function from "Functions" section of the code)

This function contains initial settings of CAN controller of LPC 2129. There are several important settings in this function. The least significant bit of C2MOD should be set to 1 before changing values in C2BTR register. After the modification is done, it should be set to zero to enable the operation. As shown in the comment of the code, the baud rate, time segments, and sampling number are to be set. As shown in Figure 1, one CAN bus bit time is longer than one clock cycle. The test was done exactly as the Figure 1, containing time segment 1 (synchronization, propagation, phase 1) and time segment 2 (phase 2) sampling only once at the Sample Point.

Figure 1[1]



Also, the acceptance filter should be selected. Acceptance filter is the filter at the receiving end, which determines which data to accept and which data to ignore. For the test, it was set as bypass since all the data transmitted were of our interest. Additionally, self-test mode can be set up so that the microcontroller can send and receive data by itself if needed.

1.1.2.c) CAN Transmit and Receive Functions
("send_CAN2" and "receive_CAN2" functions from "Functions" section of the code)

LPC 2129 has three transmit buffers and one buffer at a time can be selected to send data using transmit interrupts. For this simple test, I only used one buffer without any interrupts. Once the selected buffer is available, the transmission process should start. In this function the information about the transmitted data, i.e. data length, data type, ID length, and ID number, should be set. The code contains static data and dynamic data options, which were both tested. After the selection of data a transmit buffer should be selected, and either self-reception or transmission should be requested.
On the receiving end, CAN data reception flag from C2GSR (Global Status Register) should be checked, and the data are read to variables of the program, "CAN2_data1" and "CAN2_data2." Depending on static or dynamic data, this function should check the received data accordingly and turn on or off the LED according to the comment of the code.

1.1.2.d) Main Function
("Main Function" section of the code)

In the main function, there are some initializations needed. The CAN controller should be enabled, and some input/output ports can be determined. Also, functions such as "init_PLL" and "init_CAN2" should be initialized. After these initial functions, there is an infinite while loop that sends or receives data. Depending on the functionality of the microprocessor, one should select only one of these two functions. If "receive_CAN2" is chosen, some variables are set to check the incrementing data; on the other hand, if "send_CAN2" is chosen, it checks C2GSR and C2ICR registers to gather error information if any. The LEDs of the receiving and transmitting microcontrollers will be turned on according to the comments of the code in Appendix A.1.

1.2. Test

1.2.1. Background

One important concept to understand for data transmission is the characteristic impedance of transmission line, i.e. wire. Contrary to common assumption in introductory physics, wires are imperfect, and the imperfection needs to be accounted for data transmission via long wires. There are many termination

methods, and using a termination resister at two ends of the wire is one way to remove noise from the transmission line. The ribbon cable used in the test has the impedance of 120 ohms, and the test was performed using various resisters around and above 120 ohms.
The test was performed so that it would challenge the system and push it to its limits. Some conditions we tested were with low power high resisters (power is inversely proportional to resistance), long wires (more transmission noise), high baud rate, and nodes in between two communicating microcontrollers.

1.2.2. Testing Procedure

First task was to select the best transceiver. With 12.8m wire, DC line in between CAN wires, and close resister values, four transceivers were tested under the same condition (baud rate of 2Mbps, 4Mbps). Once the performances of the transceivers are determined, the ones that are functioning with 4Mbps baud rate are selected for the next testing. Next test was to include more range of resisters and to remove the DC line in between CAN wires. One final choice of transceiver was selected from this test.
Once the best transceiver was selected, challenging the system continued with more range of resisters, different length of wires, and high baud rates.

1.3. Result

The following table has power consumption comparisons of the four transceivers. It does not include the full data. The full data table is in Appendix A.4.

Condition: 12.8m wire, with DC line, 200 ohms (if not indicated)
Power Consumption Table

	2 Mbps	4 Mbps
MAX 3057	79.25mW	~84mW
MAX 3051	~45.22mW (150 ohms)	51.65mW
SN65VHD230	failed	failed (120ohms)
SN65VHD233	not recorded (success)	failed

The shaded portion indicates selection of the optimal components. After the first task, MAX 3057 and MAX 3051 were selected for more testing. The table below contains some previous results with some new results.

Condition: 12.8m wire
Power Consumption Table

	with DC line		without DC line
	2 Mbps	4 Mbps	2 Mbps
	150 ohms	200 ohms	240 ohms
MAX 3057	111.25mW	~84mW	95.75mW
MAX 3051	~45.22mW	51.65mW	48.02mW

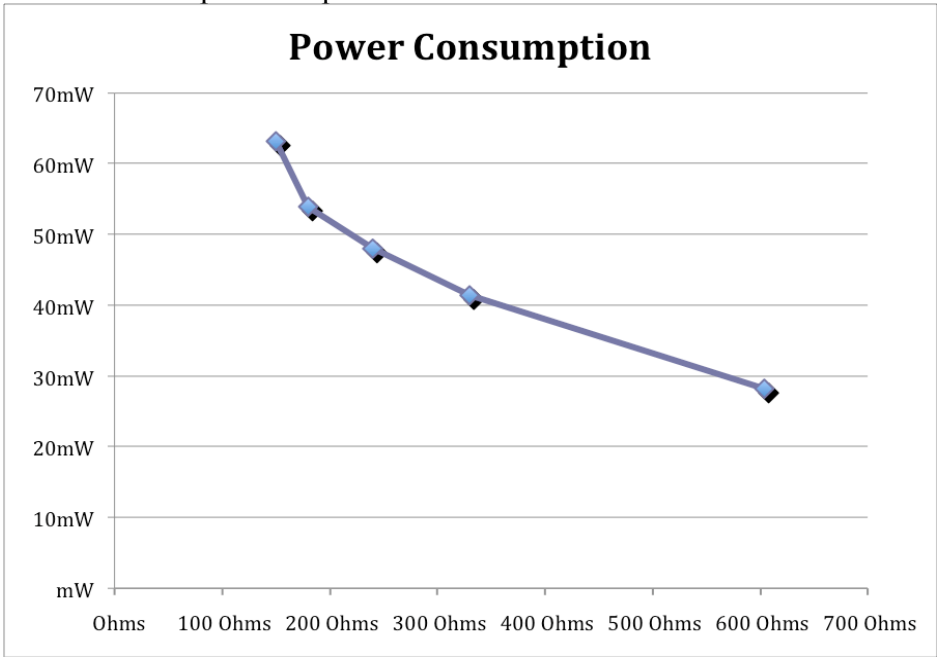
The shade portion indicates the selection of optimal component. Since MAX 3051 has low power consumption, it was selected for further testing.

Component: MAX 3051
Condition: 12.8m wire, without DC line, 2 Mbps
Power Consumption Table

Resister	Power
150 ohms	63.20 mW
180 ohms	53.95 mW
240 ohms	48.02 mW
330 ohms	41.42 mW
604 ohms	failed (28.22 mW)

The graph of the data is as follows:

Power Consumption Graph



As expected, the higher the termination resister was, the smaller the power consumption. Although the signal got noisy with high termination resister, all the signals, except for the failed case with 604 ohms, were clear enough to decipher.

Finally, MAX 3051 was tested with 4 Mbps with various resister values and wire lengths. 4Mbps testing was performed not to test the future usage but to challenge the stability of the transmission system.

Component: MAX 3051
Condition: 4 Mbps, without DC line

Power Consumption Table

	3.8 m wire	4 m wire*	5 m wire	6 m wire	10 m wire
150 ohms	58.25 mW	58.25 mW	58.25 mW	failed	not teseted
180 ohms	not tested	not tested	not tested	not tested	failed
240 ohms	45.71 mW	not tested	46.70 mW	not tested	not tested
604 ohms	not tested	not tested	failed	not tested	not tested

*4m wire with more than 2m distance to each terminating resister, i.e. it was measured between two nodes of a long wire.

The shaded portion indicates the functioning condition with maximum length of the wire. With reasonable terminating resisters (150~240 ohms), the CAN transmission system is stable up to 5 m wire.

2. H-Bridge Testing

2.1. Preparation

Jason and Haji designed the motor control board with Infineon BTS7960B (H-bridge), and Nicole prepared the components for population. I populated one H-bridge board with reflow soldering method. The board needed soldering on both sides, and soldering was successful. One of the H-bridges was noticeably tilted due to unbalanced board in the toaster oven, but it still functioned correctly. Reflow soldering with a toaster oven was still an effective method if done with caution: balancing the board, letting the board sit after turning off the oven, and cooling down the board before moving it.

2.2. Test

The motor control board was connected to 1 ohm thermal resister and received 18V with different Pulse Width Modulation (PWM) percentages. PWM level was adjusted to reach incrementing integer value of current. At each current, the maximum temperature of the component was measured.

2.3. Result

Component: Infineon BTS7960B
Condition: 18V, 20kHz PWM frequency
Temperature of the H-Bridge

Current (A)	PWM (%)	Temp (°F)	Temp (°C)
1	11.2	116	46.7
2	18.3	134	56.7
3	23.6	170	76.7
4	29.5	189	87.2
5	35.8	203	95
6	44.7	217	102.8
7	50.7	225	107.2

The H-Bridge shut down after the current reached 7 A.

Discussion

1. CAN Bus Testing

Throughout the CAN bus testing, it was clear that TI transceivers have clean signals but cannot handle high baud rates, which is actually not recommended. Maxim transceivers, however can transmit data with high baud rates although the signals in general look a bit noisier than those of Maxim transceivers. After the testing, Maxim transceiver was clearly a better choice than TI transceiver for the future electronic nervous system. However, one question still remains, which is to find the optimal termination method. This testing only used basic termination method with two terminating resisters, which consume much power. Finding a better, efficient terminating method (active termination, … etc.) remains as the follow-up task.

2. H-Bridge Testing

The H-bridge shut down at around 107.2 °C as opposed to 150 °C indicated in the datasheet. Also, it overheats earlier than H-bridge (ST Micro VNH2) used in Ranger currently, which can switch up to 10A at 20kHz. The remaining task is to test the new version of Infineon BTS7960B coming out soon, and determine its thermal characteristic.

Conclusion

1. CAN Bus Testing

MAX 3051 will be used for the electronic nervous system. Although it can transmit at 4Mbps with 5m wire, it is likely that 2Mpbs will be used in the system, which was stable with 12.8m wire and up to 330 ohm terminating resister.

2. H-Bridge Testing

It is unlikely that the current version of Infineon BTS7960B will be used in the new robot. After testing the new version, we will be able to definitively conclude the thermal characteristic of the H-bridge that will be used in the new robot.

Acknowledgements

Special thanks to Jason and Sam for the technical support regarding designing the test board, programming the microprocessor, and setting up the test.
Also, thanks to professor Ruina for the opportunity to work and learn in the lab this semester.

Citations

1. Wikipedia contributors, "Controller Area Network," *Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org/wiki/CAN_bus (accessed May 12, 2008).

Appendices

Appendix A. CAN Bus

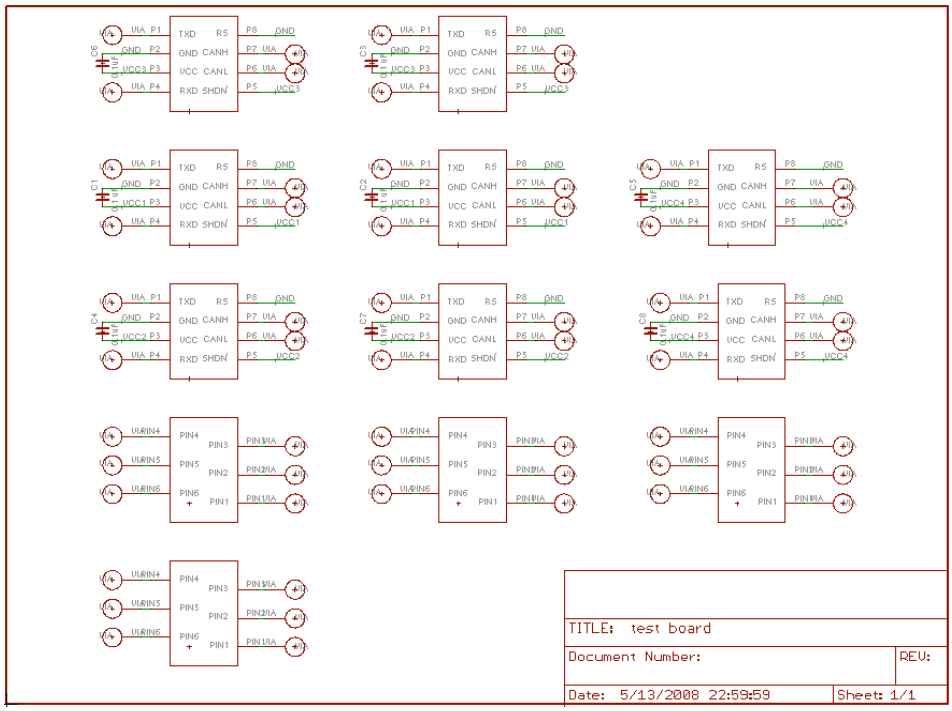
A.1
CAN Bus Test Code:

```
//*****************************************************************************
// Include Files
//*****************************************************************************
#include <inarm.h>
#include <intrinsics.h>
#include <iolpc2119.h>     // THIS IS THE MICROCONTROLLER HEADER FILE !!!!
//*****************************************************************************
//PLL USER DEFINED VALUES
//*****************************************************************************
#define CRYSTAL 10000000   // in Hertz
#define CPUSPEED 40000000  // in Hertz
#define MSEL 3
#define PSEL 1
/* !!!!PLEASE CHECK !!!!
---if it is set wrongly, microcontroller WOULD NOT RUN!---
a) 10000000 < CRYSTAL < 25000000
b) 10000000 < CPUSPEED < 60000000
c) CPUSPEED = M * CRYSTAL
c) FCCO = CPUSPEED * 2 * P
   156000000 < FCCO < 320000000
d) M = 1, 2, ..., 32
   MSEL = M - 1;
e) P = 1 , 2 , 4 , 8
   PSEL = 00 01 10 11 */
//*****************************************************************************
// Global Variables
//*****************************************************************************
unsigned long int CAN2_data1;
unsigned long int CAN2_data2;
unsigned long int CAN2_error;
unsigned long int check_c2tfi1;
unsigned long int old_CAN2_data1 = 0, old_CAN2_data2 = 0;
unsigned long int txCAN2_data1 = 0, txCAN2_data2 = 0;
//*****************************************************************************
// Initialize PLL
//*****************************************************************************
void init_PLL(void)
{
  // PLLCFG: 0 pp mmmmm where pp=PSEL and mmmmm=MSEL. PSEL=1, MSEL=4 from above.
  // PLLCFG = 0x00000023;
  PLLCFG = MSEL | (PSEL<<5);

  // PLLCON: 000000 C E C=connect, E=enable. Enable, wait for lock then C+E
  PLLCON = 0x00000001;
  // Give the connect sequence
  PLLFEED = 0x000000AA;
  PLLFEED = 0x00000055;

  while(!(PLLSTAT & 0x00000400)) ;  // Wait for PLL to lock (bit 10 is PLOCK)
  PLLCON = 0x00000003;              // Enable and Connect
  PLLFEED = 0x000000AA;
```

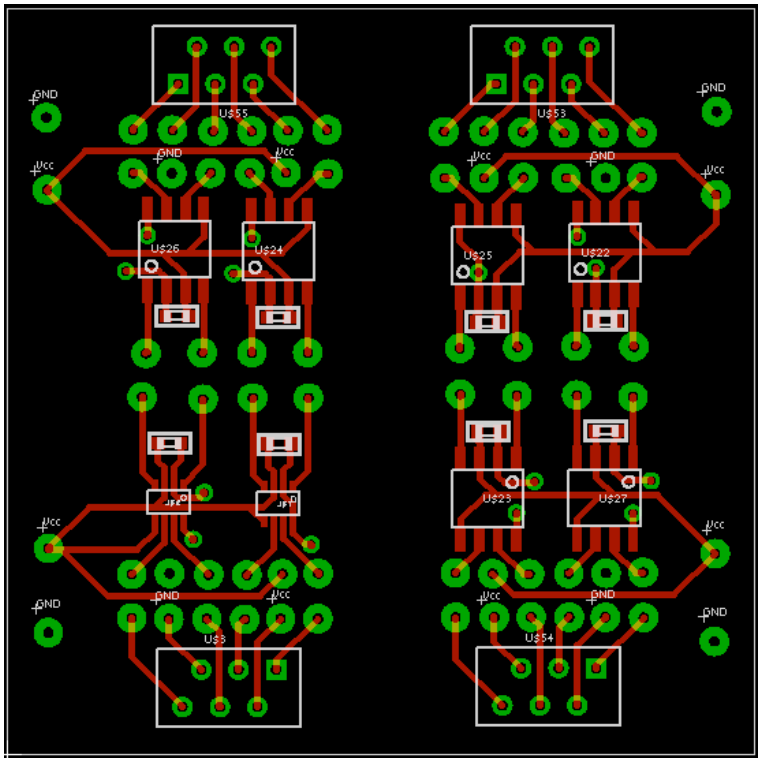
```
  PLLFEED = 0x00000055;
  VPBDIV = 0x00000001;              // VPB = processor clock
}
//*****************************************************************************
// Functions
//*****************************************************************************
void init_CAN2 (void) {
  C2MOD = 1;                    // Reset Mode
  C2CMR = 0;
  C2GSR = 0;
  C2IER = 0;
  C2BTR = 0;                    // Default Value for BTR
  C2BTR_bit.BRP = 1;            // Baud Rate = 10MHz,  Assume VPB = 40MHz
  C2BTR_bit.SJW = 1;            // Synchronization Jump Width
  C2BTR_bit.TSEG1 = 5;          // Time Segment1 = 5+1 = 6
  C2BTR_bit.TSEG2 = 2;          // Time Segment2 = 2+1 = 3
  C2BTR_bit.SAM = 0;            // Sample Once

  // Default Values
  C2TFI1 = 0;
  C2TFI2 = 0;
  C2TFI3 = 0;
  C2TID1 = 0;
  C2TID2 = 0;
  C2TID3 = 0;
  C2TDA1 = 0;
  C2TDA2 = 0;
  C2TDA2 = 0;
  C2TDB1 = 0;
  C2TDB2 = 0;
  C2TDB3 = 0;

  /* SELF-TEST MODE OPTION */
  //C2MOD_bit.STM = 1;          // Self Test Mode
  //C2MOD_bit.TM = 0;           // Test Mode

  C2MOD_bit.RM = 0;             // Operation Mode

  // Acceptance Filter: Bypass
  AFMR_bit.AccBP=1;
  AFMR_bit.AccOff=1;
}

void send_CAN2 (void)
{
    if (C2SR_bit.TBS1) {            // buffer 1 is available
      C2TFI1 = 0;                   // default values
      C2TFI1_bit.DLC = 8;           // data = 8 bytes
      C2TFI1_bit.RTR = 0;           // Normal packet, no remote transmission request
      C2TFI1_bit.FF = 0;            // ID = 11 bits
      C2TID1 = 0x1;                 // ID = 1

      /* STATIC DATA */
      //C2TDA1 = 0xAAAAAAAA;        // load 0xAAAAAAAAAAAAAAAA;
      //C2TDB1 = 0xAAAAAAAA;

      /* DYNAMIC DATA */
      C2TDA1 = txCAN2_data1;        // load incrementing numbers;
      C2TDB1 = txCAN2_data2;
```

```
      C2CMR = 0x0;                  // default
      C2CMR_bit.STB1 = 1;           // select buffer 1 and transmit

      /* CHOOSE SELF-TEST OR TRANSMISSION */
      //C2CMR_bit.SRR = 1;          // Self Reception Request
      C2CMR_bit.TR = 1;             // transmission request

      txCAN2_data1 = txCAN2_data1+1; // increment data
      txCAN2_data2 = txCAN2_data2+1; // increment data
    }
}

void receive_CAN2 (void)
{
  if (C2GSR_bit.RBS) {              // CAN data available
  CAN2_data1 = C2RDA;               // read CAN data
  CAN2_data2 = C2RDB;

    /* STATIC DATA */
    //if (CAN2_data1==0xAAAAAAAA && CAN2_data2==0xAAAAAAAA){
    //  IO1SET_bit.P1_25 = 1;       // LED ON if received correct data
    //}

    /* DYNAMIC DATA */
    if ((CAN2_data2-old_CAN2_data2!=1)||(CAN2_data2-old_CAN2_data2!=1)){
        IO1SET_bit.P1_25 = 1;       // LED ON if received incorrect data
    }

     C2CMR_bit.RRB = 1;             // release CAN data
  }
}

//*************************************************************************
// Main Function
//*************************************************************************
void main (void)
{
  PINSEL1 |= 0x14000L;     // enable CAN controller 2 (RD2 and TD2)
  IO1DIR_bit.P1_25 = 1;    // set pin 1.25 to output
  IO1CLR_bit.P1_25 = 1;    // LED OFF initially
  init_PLL();
  init_CAN2();

  //Infinite Loop
  while(1){

    /* USE THIS WHEN RECEIVING */
    //receive_CAN2();
    //old_CAN2_data1 = CAN2_data1;   // dynamic data comparison
    //old_CAN2_data2 = CAN2_data2;   // dynamic data comparison

    /* USE THIS WHEN SENDING */
    send_CAN2();
    if (C2GSR_bit.TXERR){
    CAN2_error = C2ICR;     // Capture any error information
    IO1SET_bit.P1_25 = 1;   // LED ON if there is an error
    check_c2tfi1=0;         // dummy line for a breakpoint
    }
  }
}
```

A.2
CAN Bus Test Board Schematic



A.3
CAN Bus Test Board



A.4
CAN Bus Full Data Table

	Maxim	MAX 3057	5V				
Wire Config.	Baud Rate	Resistor Value	Measured Current	Current/unit	Power	S/F	Note
with spacing	not transmitting		5.7mA	2.85mA	14.25mW	N/A	2 units
	2Mbps	150 ohm	25.1mA	22.25mA	111.25mW	S	extra unit, 12.8m wire, uneven wave
		200 ohm	18.7mA	15.85mA	79.25mW	S	extra unit, 12.8m wire
	4Mbps	200 ohm	21.3->18mA	18.45->15.15mA	92.25->75.75mW	S	extra unit, 12.8m wire
no spacing	1Mbps	240 ohm	~22mA	~19.15mA	~95.75mW	S	extra unit, inc. data, 12.8m wire
	2Mbps	240 ohm	~22mA	~19.15mA	~95.75mW	S	extra unit, inc. data, 12.8m wire
	4Mbps	240 ohm	~14.7	~11.85mA	~59.25mW	F	extra unit, inc. data, 12.8m wire, failed to receive data

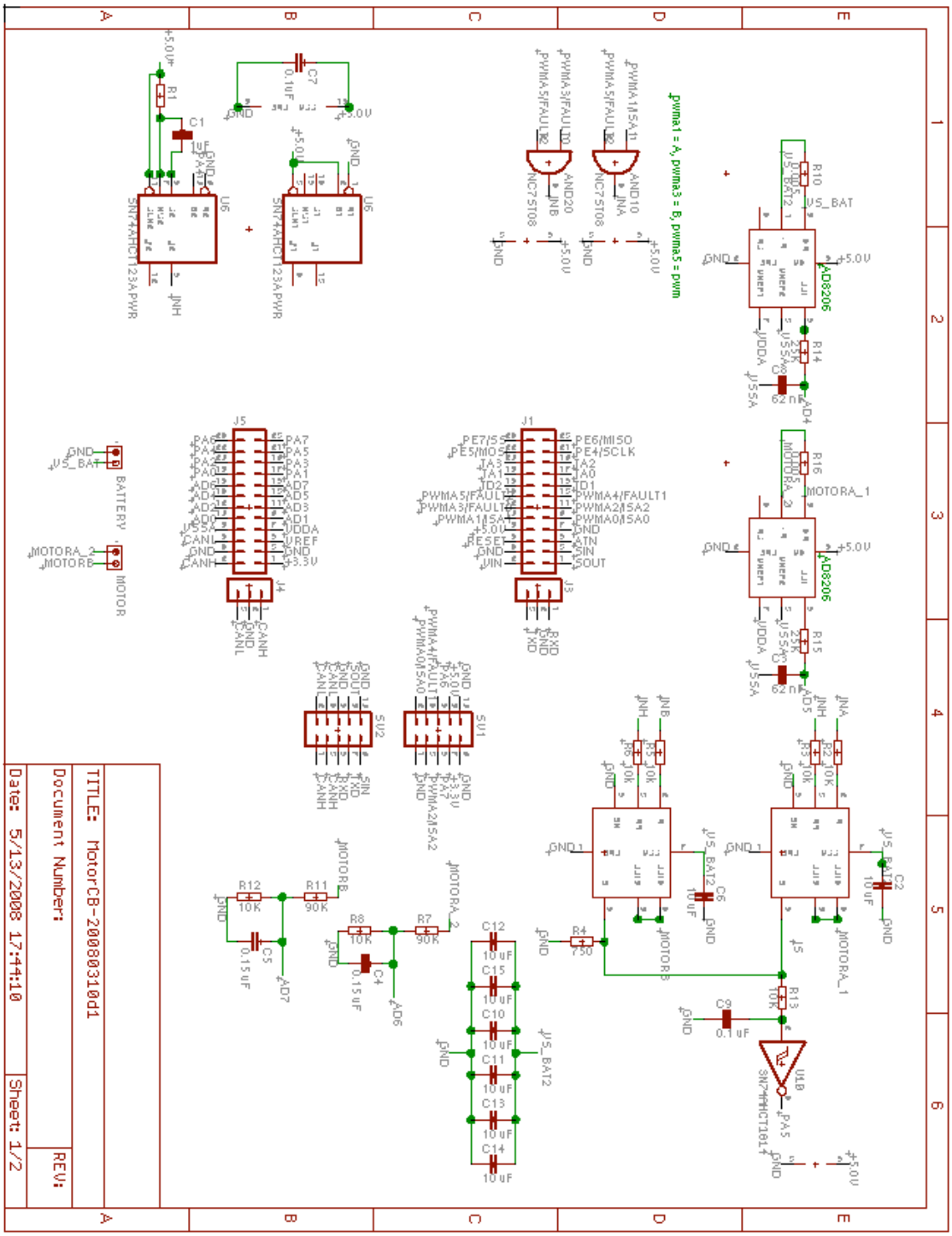
	Maxim	MAX 3051	3.3V				
Wire Config.	Baud Rate	Resistor Value	Measured Current	Current/unit	Power	S/F	Note
with spacing	not transmitting		1.7mA	0.85mA	2.81mW	N/A	2 units
	1Mbps	150 ohm	~20mA	~19.15mA	~63.20mW	S	extra unit, inc. data, 12.8m wire
	2Mbps	150 ohm	14->15.1mA	13.15->14.25mA	43.40->47.03mW	S	extra unit, 12.8m wire, no error, minor ringing
	4Mbps	150 ohm	~19.5mA	~18.65mA	~61.55mW	S	extra unit, inc. data, 12.8m wire
		200 ohm	~16.5mA	~15.65mA	~51.65mW	S	extra unit, inc. data, no error or lost packets, >5min
no spacing	1Mbps	180 ohm	~17mA	~16.15mA	~53.29mW	S	extra unit, inc. data, 12.8m wire
	2Mbps	150 ohm	~20mA	~19.15mA	~63.20mW	S	extra unit, inc. data, 12.8m wire
		180 ohm	~17.5mA	~16.65mA	~53.95mW	S	extra unit, inc. data, 12.8m wire, >5min w/o error
		240 ohm	~15.4mA	~14.55mA	~48.02mW	S	extra unit, inc. data, 12.8m wire
		330 ohm	~13.4mA	~12.55mA	~41.42mW	S	extra unit, inc. data, 12.8m wire
		604 ohm	~9.4mA	~8.55mA	~28.22mW	F	extra unit, inc. data, 12.8m wire, bad data
	4Mbps	150 ohm				F	error with 6m wire
			~18.5mA	~17.65mA	~58.25mW	S	extra unit, inc. data, 3.8m wire
						S	extra unit, inc. data, 4m(10m-6m) wire
						S	extra unit, inc. data, 5m wire
		180 ohm				F	error with 10m wire
		240 ohm	~14.7mA	~13.85mA	~45.71mW	S	extra unit, inc. data, 3.8m wire
			~15mA	~14.15mA	~46.70mW	S	extra unit, inc. data, 5m wire
		604 ohm	8mA	7.15mA	23.60mW	F	5m wire, did not receive data

	TI	SN65VHD 230	3.3V				
Wire Config.	Baud Rate	Resistor Value	Measured Current	Current/unit	Power	S/F	Note
with spacing	not transmitting		21.3mA	10.65mA	35.15mW	N/A	2 units
	2Mbps	200 ohm	30.1mA	19.45mA	64.19mW	F	extra unit, success -> errors
	4Mbps	120 ohm				F	failed

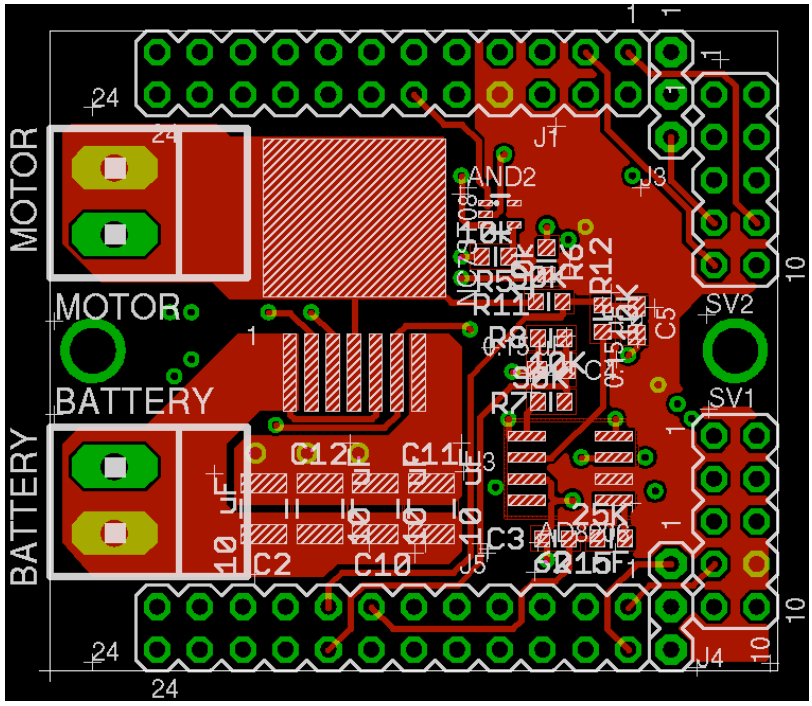
	TI	SN65VHD 233	3.3V				
Wire Config.	Baud Rate	Resistor Value	Measured Current	Current/unit	Power	S/F	Note
with spacing	not transmitting		6.9mA	3.45mA	11.39mW	N/A	2 units
	1Mbps	200 ohm	~19mA	~15.55mA	~51.32mW	S	extra unit, 12.8m wire
	2Mbps	200 ohm				S	working fine, 12.8m wire
	4Mbps	200 ohm				F	failed

Appendix B. H-Bridge

B.1
H-Bridge Test Board Schematic



B.2
H-Bridge Test Board: Top



B.3
H-Bridge Test Board: Bottom